(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 October 2002 (03.10.2002)

PCT

(10) International Publication Number
WO 02/078284 A2

(51) International Patent Classification[7]: H04L 29/00

(21) International Application Number: PCT/GB02/01428

(22) International Filing Date: 26 March 2002 (26.03.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
0107403.8 26 March 2001 (26.03.2001) GB
0112060.9 17 May 2001 (17.05.2001) GB

(71) Applicant *(for all designated States except US)*: **K TECHNOLOGIES LIMITED** [GB/GB]; 22 Dartmouth Hill, London SE10 8AJ (GB).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: **KEMP, Miles, Bonamy** [GB/GB]; 22 Dartmouth Hill, London SE10 8AJ (GB).

(74) Agent: **LANGLEY, Peter, James**; Origin Limited, 52 Muswell Hill Road, London N10 3JR (GB).

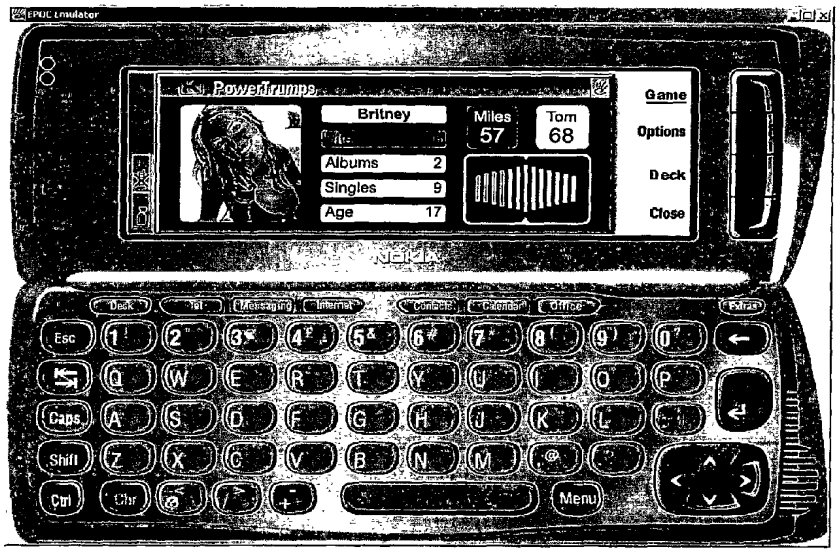
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: PEER TO PEER DATA TRANSFER BETWEEN WIRELESS INFORMATION DEVICES



(57) Abstract: Method of modifying data held in a data structure in an end user wireless information device, comprising the following steps: (a) providing a first data structure in a first end user wireless information device; (b) defining a reference which relates to data held in the first data structure; (c) sending the reference to a second end user wireless information device over a wireless bearer, the reference causing a second data structure in the second end user wireless information device to be modified in dependence on the reference, in which the reference is not processed, manipulated or used to generate any kind of related data at any intermediary server located between the first and second end user wireless information devices.

WO 02/078284 A2

PEER TO PEER DATA TRANSFER BETWEEN WIRELESS INFORMATION DEVICES

FIELD OF THE INVENTION

This invention relates to peer to peer data transfer between wireless information devices. One example is the field of multi-player games played on wireless information devices. The term 'wireless information device' includes any device able (i) to receive and send data sent at least in part over a wireless bearer and (ii) to display information. It therefore includes, without limitation, mobile telephones, communicators, smart phones, personal organizers, PCs and dedicated game playing consoles.

DESCRIPTION OF THE PRIOR ART

Typical systems for multi-player peer to peer gaming use a central server to which mobile telephones (or other kinds of wireless information device) post their game moves. It is the central server which runs the gaming application, stores recent game moves and forwards these to the other player or players; the mobile telephones typically just cache web pages received from the central server. Often, these systems are WAP based, so that only the relatively limited number of WAP enabled mobile telephones can play. Another disadvantage of these systems is that the process of posting a recent game move and receiving another player's game move can require quite significant connection time, and can therefore be costly. These problems are only exacerbated in third generation systems, such as W-CDMA and UMTS.

SUMMARY OF THE PRESENT INVENTION

In a first aspect of the invention, there is a method of modifying data held in a data structure in an end user wireless information device, comprising the following steps:

(a) providing a first data structure in a first end user wireless information device;

(b) defining a reference which relates to data held in the first data structure;

(c) sending the reference to a second end user wireless information device over a

wireless bearer, the reference causing a second data structure in the second end user wireless information device to be modified in dependence on the reference, in which the reference is not processed, manipulated or used to generate any kind of related data at any intermediary server located between the first and second end user wireless information devices.

The first and second data structures may each contain an identical or substantially similar software program and an identical or substantially similar database. By sending a reference between mirror databases in wireless information devices over a wireless bearer, as opposed to the underlying data being referenced, it is possible to significantly reduce data transfer requirements. An automated response message may be sent back to the first device, triggered by the reference.

The potency of this approach is best illustrated with reference to an actual peer to peer gaming implementation in which the object of the game is for a player to collect as many virtual 'cards' as possible on his wireless information device, with each virtual card defining data in several categories relating to an individual person or object. The first player collects on his wireless information device the virtual card of the second player if the value of the selected category for a given virtual card, at the head of the sequence stored on the device, exceeds the value of the corresponding category of the virtual card at the head of the sequence stored on the second device. For example, a virtual card could relate to a specific music artist, with categories being the number of hits, the number of albums, the number of singles and age. Hence, a first player with a virtual card relating to the music artist, say Madonna, with a particularly high number of hits could challenge another player on the 'hits' category. If the number of Madonna's hits exceeds the number of hits of the other player's top most virtual card, then the first player wins the other player's card and adds that card to his stack of virtual cards. In a game, the player with the most virtual cards is the winner.

Unlike any physical version of the game, this implementation has open-ended and unlimited gameplay. Games can be stopped and started at will – scores are ongoing, and at the end of each 2-player game the respective number of cards each player owns is

updated on a global league table, stored online. Updating from a user's device does not require any web functionality as SMS data strings can be used over a SMS to WAP/internet gateway. Players across the world can view their position on the global league, and challenge other players accordingly – whether they be based down the road or on the other side of the globe.

The need for WAP, 3G, GPRS can be eliminated with this approach since SMS, CBS or USSD may be used as the primary connectivity protocol. Card values will be stored in a database that sits on the end device. When a player selects a card to challenge another player with, the card data itself will not be sent to the other player. Instead, a string of characters in an SMS message will be sent to Player 2's device (the 'reference' in the paragraph above describing the first aspect of the invention). This string will reference the data stored in Player 2's corresponding database, for example by identifying (at least) the card and the category (e.g. card: number 16 – Madonna; category: number of hits), or simply a category identifier and the value of the category (e.g. category: hits; value: 16). The locally stored software will determine the value of this card relative to the card that Player 2 has at the top of his/her pack. A return SMS message will automatically be queried and sent to Player 1's device, informing them whether or not they have won and prompting the next turn.

These SMS messages will not be viewed by the players in native form. Rather, they will trigger the gaming engine on the device to display messages in a format consistent with the rest of the gameplay. The system uses character strings embedded in an SMS, CBS or USSD stream to reference data held in the end device. The first section of the string (e.g. a UDHI or WDP-SMS header) causes the SMS message(s) to routed a gaming application on the handset other than the conventional SMS inbox. The subsequent sections of the string reference specific content stored in a database on the end device; the software on the end device then cross-correlates this content (details of Player 1 card) against another set of content also stored on the end device (details of Player 2 card): on the basis of this, a return SMS is automatically sent to Player 1's device, which again uses an ID in the first section of the string to be routed direct to the software stored on the Player 1 device.

DETAILED DESCRIPTION

The invention will be described with reference to an implementation from K Technologies Limited of London, United Kingdom.

Overview of the process

A software program resides on a wireless information device - whether PC, phone, smartphone, PDA, or any other electronic device. It contains a database in which a variety of data is stored. This data is "tagged" - i.e. it can be referenced by a character or string of characters. A typical reference string might be:

6969696969 - PTHORR1 - 00026 - 00003

This, in explanatory terms, might represent:

Communication ID - Application involved - Field value A - Field value B

A more complex reference string is described in a later section of this specification, although its basic operation is similar. Content within the database stored on a first wireless information device, Device 1, is selected by User 1. The character string that references the selected content is sent as a data stream via a wireless network such as a GSM network, in the form of WDP-SMS or any other wireless data format, to another device or number of devices (Device 2 - Device X). This character string is received on the device or devices to which it has been sent - in this example we will assume one device, Device 2. The same or equivalent software program will need to have been pre-installed on the recipient device or devices, or some other software program that is capable of recognizing incoming data from an application that uses these character strings. The Communication ID contained in the incoming data stream automatically indicates to software on Device 2 the nature of the incoming signal, and the character string is routed to the specific application involved. Fields A and B (and any other field or fields involved) are referenced in the database stored on Device 2, and the software on Device 2 cross-correlates this data against a separate set of field data also stored on Device 2.

Based on this cross-correlation, a change to the database value or values on Device 2 might be made, and a data message may also be relayed back to Device 1 which may trigger a change to the database value or values on Device 1. A data message might also be sent to any number of other devices from Device 2, prompting an equivalent software-based evaluation process on these devices to that described above.

This process, which can use low-bit-rate wireless data formats such as SMS as its transfer mechanism, enables peer-to-peer multiplayer gaming on wireless devices without the need for an internet/server-based system as is generally employed in WAP gaming.

Potential games include:

Card trading games, where cards are won or lost according to their respective values. Basic gameplay employs the process described above as follows:

- User 1 selects a value on their top card
- this is relayed to User 2's device (Device 2)
- the pre-installed software on Device 2 checks the card value referenced by the character stream it has received from Device 1 against the relevant card value on User 2's "top card"
- If the value of User 2's top card is greater than User 1's, the number of the relevant card held on Device 2 is increased by 1 and the number held on Device 1 is decreased by 1. The latter amendment is made by an automatically-generated return data message, which is relayed back from Device 2 to Device 1
- If the value of User 2's card is less than User 1's, the opposite amendment is made.

Different card games may be based on different subjects, such as:

- Music artists (where field values relate to values such as the number of hits, number of albums, age etc.)
- Dinosaurs (where field values relate to values such as era in which the creature lived, height, weight, length etc.)

- Racing Cars (maximum speed, etc.)
- Adult entertainment (Breast size, etc.)

Competitive multiplayer character-based games, where the respective values of a visually displayed character or characters (such as physical strength, agility, weight, etc.) are stored in similar database format to the card trading game outlined above, but with a richer graphical user interface used to interpret and display this data. Database values may be dynamic, i.e. capable of changing over time. A kung-fu fighter, for example, might be "trained" by User 1 on Device 1 to improve his/her skill values, before being "sent into combat" against a character stored on Device 2.

Peer-to-peer data dissemination

The process described can also be used as a cost-effective peer-to-peer means of making changes to software and data stored on a large number of devices - without the need for any form of internet connectivity as is conventionally used for product upgrades etc. It may be that a business needs to upgrade some of the data stored on devices used by its employees. In many cases, it may be more cost-effective to use the peer-to-peer dissemination process described above, rather than send a message to every individual employee from the central source. For example, a change to the sales forecasts for a product, or an increase in the number of staff employed by head office, might be relayed to all employees in this manner, with individual devices "talking" to each other until consistency has been achieved across all devices.

Card Trading Game Example: PopSwap

PopSwap™ is a dynamic software-based version of the well-known Top Trumps™ card swapping game. Users compete to win each others cards, based on the respective values assigned to them. Unlike "physical" Top Trumps, where a game is not completed until all the cards have been won or lost, PopSwap games can be terminated at any time and any player can win a potentially infinite number of cards. Scores are updated to a central online league table, transforming the game concept from a self-contained one-off model to a dynamic, globally-interactive model.

Delivery mechanism

The PopSwap gaming application, the K-engine, will be stored locally, on the user's PC, PDA, or mobile phone. In this respect it differs from the standard WAP gaming model, where games are accessed via a remote server – a bandwidth-heavy and cost-intensive solution, which prevents LAN gaming via Bluetooth and other protocols. The PopSwap user will either (a) download a copy of the software from the PopSwap website (on PCs and other open-RAM, internet-enabled devices), or (b) use a pre-installed version where agreements with handset manufacturers have enabled this. Option (b) may require the PopSwap engine (the K-engine) to be built into handset ROM or SIM card.

Platform requirements

PopSwap should be playable on and between the following platforms:

Platform	**P2P Communication**
PC	Fixed-line internet, Bluetooth, IRDA
Mobile phone	SMS, Bluetooth, IRDA
Windows CE PDA (HP, Casio etc.)	SMS, Bluetooth, IRDA
Symbian PDA (Psion, Nokia, Ericsson etc.)	SMS, Bluetooth, IRDA
Palm OS PDA (Palm, Handspring)	SMS, Bluetooth, IRDA

Each platform is likely to require a different software build – both front-end and back-end. However, a consistent communication protocol, based on SMS (for long-range) and iRDA/Bluetooth (for short range), will enable different formats to play against each other without problems.

Functionality

Perhaps the best way to describe this is to give a step-by-step view of a typical gameplay sequence.

(i) **Player 1 selects "Send challenge" to challenge another player**

- Mobile phone will send invite via an SMS trigger to a stipulated phone address (which the user will have to enter) or locally via iRDA.

- Internet-enabled user can either (a) log on to the online league table and issue a challenge to any player, or (b) issue a challenge to a buddy-list stored locally.

(ii) **Player 2 accepts/rejects offer**

- If accepts, gameplay mode is initiated on both units.
  - Both packs are shuffled
  - A starting player is randomly selected, based on a "heads or tails" choice by the challenger
  - The game begins.
- If rejects, rejection notice is relayed back to Player 1.

(iii) **Player 1 selects a field attribute from their top card,** e.g. "Physical Strength 86". This selection is relayed to Player 2. The full card data is not sent – this would be slow & bandwidth-intensive. Instead, a character string is sent that references Player 1's card & field – details of which are of course also stored in Player 2's content database. An example string may look like this:

6969 HOR 0003 0001

6969 - tells Machine 2 that the incoming message relates to K-engine

HOR – references the "Horror" pack of cards

0003 – Card 3 in the pack (e.g. "Circus of Death")

0001 – Field 1 of this card (e.g. "Physical Strength")

(v) **Machine 2 interrogates its database** to find out what Circus of Death's Physical Strength value is. This is then correlated against the field value for Player 2's card (e.g. "Devil Fiend"), to see which is greater.

(vi) **If Circus of Death > Devil Fiend,**

- the reference string for Player 2's card is sent to Machine 1, and the number of Devil Fiends held in Player 1's database is increased by 1.
- Both players are notified, and are able to view their opponent's card if they wish.
- Both cards are sent "to the back of the pack", i.e. appearance number changed to 39 & 40.
- It is Player 1's turn again. Return to step (iii) with the next card in the pack

If Circus of Death < Devil Fiend,

- the number of Circus of Deaths held by Player 2 increased by 1, and gameplay control shifts to Player 2.
- Again, both players are notified and given option of viewing opponent's card.
- Again, both cards are sent to the back of the pack.

(vii) **Game ends when**:

- 5 minutes elapse after last exchange of cards
- One player selects "End this game"

In either case,

- both players are notified of the game's end via text message.
- the number of cards each player holds is sent to the online league table, where their positions in the league are updated. Number of turns played per game is also provided, so that gamers checking the league table can be kept alert for sneaky gaming techniques.

Additional functionality

More sophisticated features may be included, such as:

- **Invite new player** – sends a text message/email to a friend, inviting them to download PopSwap and providing them with the download URL.
- **Create ID** – enables user to create a number of user IDs – so that if they are doing badly on one ID, they do not simply lose all their cards and grow disenchanted.
- **1-Player game** – enables user to play against the computer/phone. Scores here do not get added to the league, nor do they "roll over" from one game to the next.
- **Get new games** – direct link to download centre for other K-Engine-compliant games. N.B. This means the K-Engine database(s) should be capable of handling more than one game, although only 1 can be played at any time.
- **Check for upgrades** – enables user to see if new cards have been created that they can add to their pack (for a fee). N.B. This means it should be possible to

add whole new entries to the K-Engine database, alongside existing entries.

Architecture, design and implementation

This section outlines the technical architecture, design and implementation of PopSwap (phase 1). PopSwap can be considered to consist of the following key modules:

1. The user interface ("UI"), responsible for all interactions with the game player; this includes displaying appropriate information before, during and after play; producing a clear, consistent and navigable model for menus and so forth; and gathering user input at appropriate points (for example, when a player is required to select an attribute);

2. The game engine ("GE"); responsible for keeping track of all aspects of game state, moving between states, the reading and writing of information to and from storage, and insuring consistency of all data;

3. The communications engine ("CE"), providing a means of realising multi-player functionality; allows for abstracted communication across a variety of different bearer networks, with consideration for the characteristics (functional and commercial) of the individual networks;

4. An overall shell ("shell") responsible for starting the application and firing up the other modules as and where necessary.

Wherever possible, PopSwap has been designed to keep these modules as separate as possible. The thinking here is that the prototype could be easily demonstrated on a different device by using the same CE and GE, and only developing a new UI (and possibly shell). Likewise, new bearers for the CE (e.g. Bluetooth) could be developed without having an impact on the UI.

Class hierarchy

The hierarchy of Java classes for the Nokia 9210 implementation within PopSwap can

be summed up in **Figure 1**.

com.ketch.trumps.ui.Display

Display is responsible for maintaining the main screen panel, where most of the UI of PopSwap is located. The exact contents of this panel are dealt with in greater depth in the "user interface" section, but typically they would include a wallpaper (if no game is in progress), or an image of the current card, details of the attributes and title of that card, and an indication of current attribute choice and game status (should a game be in progress). Display will make use of double-buffering and other techniques to ensure that screen redraws are carried out smoothly.

Class variables include:

```
private com.ktech.trumps.engine.Game game;
```

Public methods include:

```
public void redraw();
```

com.ketch.trumps.ui.KeyHandler

KeyHandler deals with key-presses from the user and fires off appropriate events. Typically, key-presses are only listened to when a game is being played (e.g. to move the current choice of attribute up and down using cursor keys, and to select it using the "enter" key). It implements the java.awt.event.KeyListener interface. KeyHandler does not have any class variables.

Public methods (specified by the KeyListener interface) include:

```
public void keyPressed(KeyEvent e);
public void keyReleased(KeyEvent e);
public void keyTyped(KeyEvent e);
```

com.ketch.trumps.ui.PopSwapCBAHandler

PopSwapCBAHandler sets up and handles interactions that involve the Command Button Array (located on the right-hand side of the main display, on the 9210 emulator). Extending com.symbian.devnet.crystal.awt.CBAHandler, it handles the switching of

menus as game-play proceeds and the launching of events, dialog boxes, or game states as a result.

com.ktech.trumps.engine.Card

Card encapsulates all the information needed about a single card in a pack of cards.

Class variables include:

```
public String Name;
public String Labels[];
public int Values[];
public String Picture;
```

Name is the name of this card; Labels[] an array of labels for the attribute values of this card, and Values[] an array of the actual values themselves. These two arrays are necessary to cope with circumstances where attribute units (and thus their numerical values) might vary, but the attributes have to be comparable (for instance, comparing "500kg" with "2 tonnes"). Picture is the filename of an image for this card.

com.ktech.trumps.engine.Pack

Pack describes a single pack of cards.

Class variables include:

```
public String Name;
public String UID;
public int NumberAttributes;
public int NumberCards;
public String Attributes[];
public Card Cards[];
```

Name is the name of this pack of cards (for example, "Buffy Trumps"); UID a 16-character string uniquely identifying this pack; NumberAttributes, as its name suggests, contains the number of attributes which each card in this pack contains, and NumberCards is the number of cards in the pack. Attributes[] is an array containing the names of these attributes, and Cards[] an array containing all the cards in this pack.

com.ktech.trumps.engine.Player

Player holds all the information about a single player.

Class variables include:

```
public String Name;
public String UID;
public int Deck[];
```

Name is the name of this player; UID a 16-character string uniquely identifying him or her; and Deck[] is an array of integers, detailing which cards the player currently holds (and in what order). An array has been chosen over a Vector (and integers stored rather than Cards) for reasons of efficiency.

com.ktech.trumps.engine.Game

Game encapsulates all the necessary information needed to represent a single game of PopSwap at any one time.

Class variables include:

```
public String UID;
public Player PlayerOne;
public Player PlayerTwo;
public Pack GamePack;
public int State;
public Date LastStateChange;
```

UID is a 16-character string uniquely identifying this game; PlayerOne and PlayerTwo are the two participants; GamePack the pack of cards being used for the purpose of this game; State the current state of the game; and LastStateChange the time when the last change was made to the game state. This last variable is used to help implement time-outs and the like, where if (for example) 60 seconds have elapsed since a move was requested, a warning might be given.

A number of constants are also defined, to represent different game states (as per the scoping document, PopSwap game-play is simple enough to be implemented as a Finite State Machine):

```
public static final int START = 0;
public static final int CHALLENGE_SEND = 1;
public static final int CHALLENGE_DECLINED = 2;
public static final int COMM_ERROR = 3;
public static final int FINISH = 4;
public static final int MOVE_CHOOSE = 5;
public static final int MOVE_SEND = 6;
public static final int AWAIT_MOVE = 7;
public static final int HANDLE_MOVE = 8;
public static final int LOSE_ROUND = 9;
public static final int WIN_ROUND = 10;
public static final int LOSE_GAME = 11;
public static final int WIN_GAME = 12;
public static final int RECEIVE_CHALLENGE = 13;
public static final int DECLINE_CHALLENGE = 14;
public static final int CONFIRM_FINISH = 15;
public static final int SELECT_START = 16;
```

Definitions: the following are the mandatory, standard terms

Game

Challenge

Pack

Deck

Card

Player

Turn

Item/Attribute/Value/

View deck

League

Status message

Dialog

User Interface

Nokia 9210 interface

The following section describes the Nokia 9210 implementation of the PopSwap interface. The 9210 has four command buttons which are assigned context-sensitive actions for OK'ing dialogs, selecting items and so on. **Figures 2 and 3** show the the 9210 device for the PopSwaps (music artist) implementation of PopSwap. **Figure 4** shows the Handspring implementation. A single card is shown in each Figure with the picture of a music artist and 4 categories of information: Number of: Hits, Albums, Singles and Age. Figure 2 shows how scores can be shown in the user interface (with Miles having 58 cards and Tom, leading with 67 cards). A visual indicator displays these scores and also graphically represents the size of each players pack by a series of parallel lines. In addition, an instant messaging chat service (not shown) can be included in the interface, allowing players to send messages to one another as part of the gameplay experience. **Figure 5** shows the simplified interface possible with a GSM implementation.

The following table indicates all the actions assigned to the command buttons, during all stages of gameplay.

Application command button actions

This table shows the softkeys (Command Buttons) context at each point in the game, it does not include the actions for the command buttons which are linked to in-game dialogs which are described below.

Game state	CB 1	CB 2	CB 3	CB 4
No game in progress. The topmost card is the pack "cover" graphic with the name of the pack, logos and branding, etc. **Challenge** opens the Issue Challenge dialog **View pack** opens the pack browsing **Close** quits the application	Send challenge	View pack	Options	Close
Game in progress. **Use attribute** is active when it is the user's turn to select an attribute from the current card. It is inactive when an attribute has been selected and a response from the other player is being awaited. **Info** reveals the information about the current card if the picture is currently displayed (as does the **Tab** key) **Picture** reveals the picture for the current card if the textual information is being displayed (as does the **Tab** key) **Options** opens the Options dialog **End game** opens the End Game? dialog	Select	View pack	Options	End game

Command buttons are context sensitive; currently selected card item has a red highlight;

current player is indicated by white highlight. Players can browse all the cards in their pack

Communications Protocols

We anticipate that, despite the range of potential CEs, a common protocol will be implemented across them.

Due to the nature of some initial CEs, which are suitable for transporting alphanumeric data (notably SMS), the protocol will be text-based. The limitations of SMS mean that the maximum size of any possible message in the protocol must be 160 characters (ideally less, to allow for future expansion).

A "protocol identifier", used to allow future protocols to detect older versions and remain backwards compatible, precedes every message sent. This protocol id will be "KTTP/1.0" for this revision. Elements within the messages will be separated using "carriage return" characters. Where not otherwise specified, characters within messages are case insensitive (except for names and unique identifiers).

This protocol, illustrated by the **Figure 5** flowchart, will consist of the following possible messages:

Send challenge

Sent when a player challenges another player to a game, this message consists of:

1. A message identifier: the word "CHALLENGE"
2. A unique identifier for this game, generated at this point (16 characters)
3. The challenging player's unique identifier (16 characters)
4. The unique identifier for the pack of cards the challenger wishes to play with (16 characters)
5. A random positive integer (0-65535), used to determine who goes first in the game (the sender highest of this integer and the one in "Accept challenge" wins)
6. The number of cards remaining in the challenger's deck
7. The challenger's name

A challenge message might therefore read (with <cr> denoting a carriage return):

KTTP/1.0<cr>
CHALLENGE<cr>
AL29FN49SH10SM47<cr>
0PN39DN8WMLUN249<cr>
P1P138GHKS8FB28C<cr>
10245<cr>
12<cr>
Tom Hume<cr>

This message would total 88 characters.

Accept challenge

Sent in response to a "Send challenge" message, when the challenged player wishes to play. Consists of:

1. A message identifier: the word "ACCEPT";
2. A unique identifier for the game (16 characters); this should be the same as the game identifier received in the "Send challenge" message;
3. The challenged player's unique identifier (16 characters);
4. A random integer, used to determine who goes first in the game (see "Send challenge" above)
5. The number of cards remaining in the challenged player's deck;
6. The challenged player's name;

So, an example message (sent in response to the example in "Send challenge") might be:

KTTP/1.0<cr>
ACCEPT<cr>
AL29FN49SH10SM47<cr>
8HQ92M56SJ2L18DN<cr>

24042<cr>
16<cr>
Jay Gooby<cr>

Making a total of 69 characters.

Decline challenge

Sent in response to a "Send challenge" message, when the challenged player does not wish to play. Consists of:

1. A message identifier: the word "DECLINE";
2. A unique identifier for the game (16 characters); this should be the same as the game identifier received in the "Send challenge" message;

An example message from a player declining the challenge sent in "Send challenge" (above) might be:

KTTP/1.0<cr>
DECLINE<cr>
AL29FN49SH10SM47<cr>

"Decline challenge" messages therefore have a fixed length of 34 characters.

Send move

The most commonly used message, this is used to report a single move in the game. Consists of:

1. A message identified: the word "MOVE";
2. A unique identifier for the game (16 characters); this should be the same as the game identifier received in the "Send challenge" message;
3. The number of the card being played, within the pack being used for this game (numbered 0 to *n*);
4. The attribute number being played, on this card (numbered 0 to *n*);

An example message, in keeping with the other examples shown here, might be:

KTTP/1.0<cr>
MOVE<cr>
AL29FN49SH10SM47<cr>
5<cr>
1<cr>

This one contains 35 characters.

End game

Used to alert the other player that the game ended; consists of:

1. A message identifier: the word "END";
2. A unique identifier for the game (16 characters); this should be the same as the game identifier received in the "Send challenge" message;

An example message, in keeping with the other examples shown here, might be:

KTTP/1.0<cr>
END<cr>
AL29FN49SH10SM47<cr>

"End game" messages are therefore always 30 characters long.

Report score

Used by both players of a game, when the game has ended, to report their current score (and the game status) to the central high score table; consists of:

1. A message identifier: the word "SCORE";
2. A unique identifier for the game (16 characters); this should be the same as the game identifier received in the "Send challenge" message;
3. The unique identifier for the pack of cards used to play this game (16 characters);
4. A unique identifier for this player (16 characters);

5. The name of this player;
6. The number of cards this player currently holds;
7. The number of cards this player won during the game;
8. The number of cards this player lost during the game;
9. The overall winner of the game (1 if it was this player, 0 if it was the other);

For example, should the challenger have won in our example, the following message would be sent:

KTTP/1.0<cr>
SCORE<cr>
AL29FN49SH10SM47<cr>
P1P138GHKS8FB28C<cr>
0PN39DN8WMLUN249<cr>
Tom Hume<cr>
22<cr>
30<cr>
20<cr>
1<cr>

This message would be 90 characters long.

Receive rank

This message is sent by the "high score" subsystem back to any player who sends in a "Report score" message. It confirms their current ranking in the league table for their deck, and consists of:

1. A message identifier: the word "RANK";
2. A unique identifier for the player (16 characters);
3. An integer indicating their rank in the league table

An example message might be:

KTTP/1.0<cr>
SCORE<cr>
0PN39DN8WMLUN249<cr>
102<cr>

This message would be 36 characters long.

Data Exchange

This section outlines the technical process by which data exchange takes place. It refers specifically to an implementation for the Nokia 9210 Communicator, using a Java implementation on the EPOC 6.0 OS. However, the application could also be run on a conventional (ie non-WAP) GSM mobile phone, using API calls within the proprietary phone programming environment.

Devices participating in a game of PopSwap communicate as follows: Assume two devices, A and B. A has a message to send to B; this might be one of several types of message (e.g. a challenge, a response to a challenge, the details of a card and choice of attribute, etc.). The exact type of message is determined by what state the game is in, and is not relevant to this discussion. For our purposes, messages can be divided into one of two types: outgoing (sent to the other player), or incoming (received from the other player).

1. Outgoing messages

The PopSwap application composes the message into a single text string, suitably formatted as described above. PopSwap then determines the recipient of the message, which consists of a telephone number (the number of the other mobile device), and a WDP port number (PopSwap uses a standard hard-coded port number). Classes within the Java Telephony API (JTAPI) are used to combine the text string with these two pieces of information, into a "WDP-SMS" message, and send it. The JTAPI handles the interfacing between PopSwap and the EPOC operating system services which actually send the message.

2. Incoming messages

The device receives an SMS message, notices that it is not "raw" SMS, but contains WDP-SMS protocol, and determines which WDP-SMS port the message is being sent to. It then looks for applications which are currently listening on that port; PopSwap will be the only such application, so the message is forwarded to a "listening thread" which PopSwap activates whenever it expects to receive a communication. This "listening thread" receives the complete KTTP message (which has been transparently disassembled from the WDP-SMS message by JTAPI) and decomposes it into its differing fields, acting appropriately. For instance, should a PopSwap device receive a message containing the details of a move, it will go through the following steps:

1. Examine the "Game UID" provided with this message, to ensure that the message applies to the game currently being played; game UIDs are exchanged privately between players when a challenge is made, and this prevents the "hijacking" or disruption of game sessions by malicious third parties.
2. Record the "attribute ID" to know which attribute is being played.
3. Look at the card ID and use this to reference a card in its deck database (and the attribute ID to reference an attribute within this card).
4. If a response is required (i.e. it is the other players' move, and we need to inform them of which card we have next) it is sent, as per section 1 above.

We therefore have our communications structured into layers (as per the OSI model, a standard way that networks are explained):

- PopSwap itself developed by us
- JTAPI provided with the Java implementation for EPOC
- EPOC services provided by the EPOC OS
- Communication with hardware provided by the hardware manufacturer
- The phone network itself provided by the network.

Extensions to the core concepts

The following are possible extensions to the implementation described above.

1. "Send challenge" concept

Summary

A form of data delivery and receipt that enables a standardized, broadcast data message to be interpreted in a variety of ways depending on the software installed on the receiving device.

Example: When a K Technologies Games user (user/device 1) sends a challenge to another phone number, an automated data string is sent to the other device (user/device 2). This message is interpreted differently according to whether or not device 2 has K Technologies software installed on it. If compatible software is present, the automated accept/decline challenge cue is initiated; if it is not, the data message sent is reinterpreted and displayed as an invite for user 2 to download K Technologies software for themselves. A URL may be given to enable them to do so directly.

The data string may be sent in SMS format, as follows:

- 69696969 – KPOPT0001 – MILESK – 0000142 – MILESK invites you to try out PopSwap, the new multiplayer game for your phone or PDA. Visit www.network-k.com for info and downloads.
  (a) if user 2 has K Technologies' PopSwap installed on their device, PopSwap is initiated, and the first 4 sections (Generic K Tech tag, Game tap, username, number of cards held) are routed accordingly. The section of the SMS after hyphen (–) 4 is automatically discarded.
  (b) if user 2 does not have PopSwap installed, the message is handled and displayed as a standard SMS message, and the user has access to the promotional message in full.

Potential applications:

- **Wireless multiplayer games** in which a challenge from one user to another is used to initiate gameplay
- **Promotional messaging** where the nature of the promotional message may vary according to data held on the end device – a number of supermarket loyalty

card holders, for example, could be notified of different offers according to the data held on their device(s), even though one standard message was broadcast to all users

- **News bulletin messaging** where a single broadcast message could result in user-customized information being displayed onscreen according to the preferences made by the user and stored locally on the user's device

2. "League Update" concept

Summary

Automated data update system between a client device or devices and a remote data storage location, e.g. server via wireless connection without utilizing http or other internet protocols.

Example: When a K Technologies gaming session between two devices is ended, an automated message is sent from both devices to a remote server via SMS or other wireless data format. These messages encode the final score of both players.

- When these messages reach the server, they are inputted into a "league table" ranking some or all K Technologies games players by score or status. An automated reranking calculation is carried out by the server based on the new data that has just been received, and an automated message containing the revised league positions of either or both players is relayed back to the device or devices from which the messages were received. Again, this return message may be sent in SMS or any other wireless data format.

Potential applications:

- **Wireless multiplayer games** in which an element of ongoing league-based competition is involved
- **Academic examinations** in which the rankings of individuals scholars is automatically calculated and relayed back to them on completion of their exams

- **Business applications** in which individual usage of various locally held applications is monitored and settings on the client device automatically adjusted as a result

3. "Auto-challenge" concept

Summary

A system between a client device or devices and a remote data storage location (e.g. server) via a wireless connection whereby users are matched with other users recorded on the server according to the values in data fields encoded in the message or messages sent from client to server, and an automated connection is made between two or more client devices on the basis of these values.

Example: A K Technologies games user may want to play a multiplayer game but not be acquainted with other users of the game or games in question. By selecting "auto-challenge" (or some other appropriately named option) on their device, an automated message is sent from user 1's device to a remote server (via SMS or any other wireless data format), detailing the user's status (e.g. number of cards held, number of games played, field values of character selected, etc.).

- User 1 may also define the difficulty level or status of the user they wish to challenge. For example this may be ranked on a scale of 1 to 5 – from "beginner" through "intermediate" to "pro". This data will also be encoded in the message sent from device 1 to the remote server.

- Incoming data from user 1 is received by the remote server, and based on the values encoded, an automated selection is made of another user recorded on the system whose values are in accordance with the criterion or criteria sent from user 1. An automated message is then dispatched to the selected user 2, informing them of the challenge and asking them if they would like to accept or decline.

- If the challenge is accepted, gameplay is initiated in the normal manner and an automated amendment to the data on the server (e.g. registering user 2's propensity to accept auto-challenges) may be made. If the challenge is declined, a message is relayed back from device 2 to the server and either (a) a further selection may automatically be made, (b) a notification message may be sent back to user 1, or (c) both re-search and notification message may automatically be generated. Again, an update or amendment to the data held on the remote server may be made.

Potential applications:

- **Wireless multiplayer games**
- **Dating services**, where a user defines their characteristics and an automated match is made with a suitable potential partner or partners

4. "Dynamic values" concept

Summary

System whereby regular data updates are sent from a remote data storage location (e.g. server) to a number of client devices via SMS or any other wireless data format, updating or amending field values stored in databases on the client device. These updated data values may then be referenced by data sent in the form of character strings between individual client devices.

Example: PopSwap is a wireless games concept, conceived and owned by K Technologies, in which bands and musicians (i.e. "act") are traded between players according to various data values (e.g. number of Top Ten Hits, number of albums, etc.) pertaining to each act.

- If left static, the data relating to each act would rapidly become obsolete, as new chart hits each week changed the true values for each act. Therefore a regular transmission of revised data fields (perhaps immediately after the weekly publishing of the billboard charts) would ensure that the content of the game

remained precisely up to date, and that the game retained a dynamic, up-to-date dimension.

- This revised data would then be referenced in all relevant data exchanges between client devices.

Potential applications:

- **Wireless games** such as the one outlined above (PopSwap)
- **Peer-to-peer business applications** in which time-sensitive information such as inventory levels and stock price are stored on the client device or devices.

CLAIMS

1. Method of modifying data held in a data structure in an end user wireless information device, comprising the following steps:

(a) providing a first data structure in a first end user wireless information device;

(b) defining a reference which relates to data held in the first data structure;

(c) sending the reference to a second end user wireless information device over a wireless bearer, the reference causing a second data structure in the second end user wireless information device to be modified in dependence on the reference, in which the reference is not processed, manipulated or used to generate any kind of related data at any intermediary server located between the first and second end user wireless information devices.

2. The method of Claim 1 in which the first and second data structures each contain an identical or substantially similar software program.

3. The method of Claim 1 or 2 in which the first and second data structures each contain an identical or substantially similar database.

4. The method of Claim 1 in which the reference is coded as one or more SMS format messages, and data encoded within the or each SMS message prompts the reference to be routed to the data structure instead of being treated as a standard SMS message.

5. The method of Claim 1 in which the wireless bearer is one of the following wireless bearers:

(a) GSM

(b) UMTS

(c) W-CDMA

(d) Short range wireless

6. The method of Claim 1 in which the wireless bearer is GSM or GPRS and the reference is coded as one or more SMS or CBS or USSD format messages.

7. The method of Claim 1 in which the wireless bearer is UMTS or W-CDMA and the reference is coded as one or more USSD format messages.

8. The method of Claim 1 comprising the further step of the reference causing an automatic response to be returned to the first wireless information device from the second wireless information device.

9. The method of any preceding Claim forming the data transfer mechanism used to enable multi-player games to be played between users of the first and second wireless information devices.

10. The method of Claim 9 in which the game is a card trading game.

11. The method of Claim 10 in which the reference sent from the first wireless information device represents a value of or associated with a card.

12. An end-user wireless information device comprising a data structure which is programmed to be modified according to the method of any of Claims 1 – 11.

Figure 1

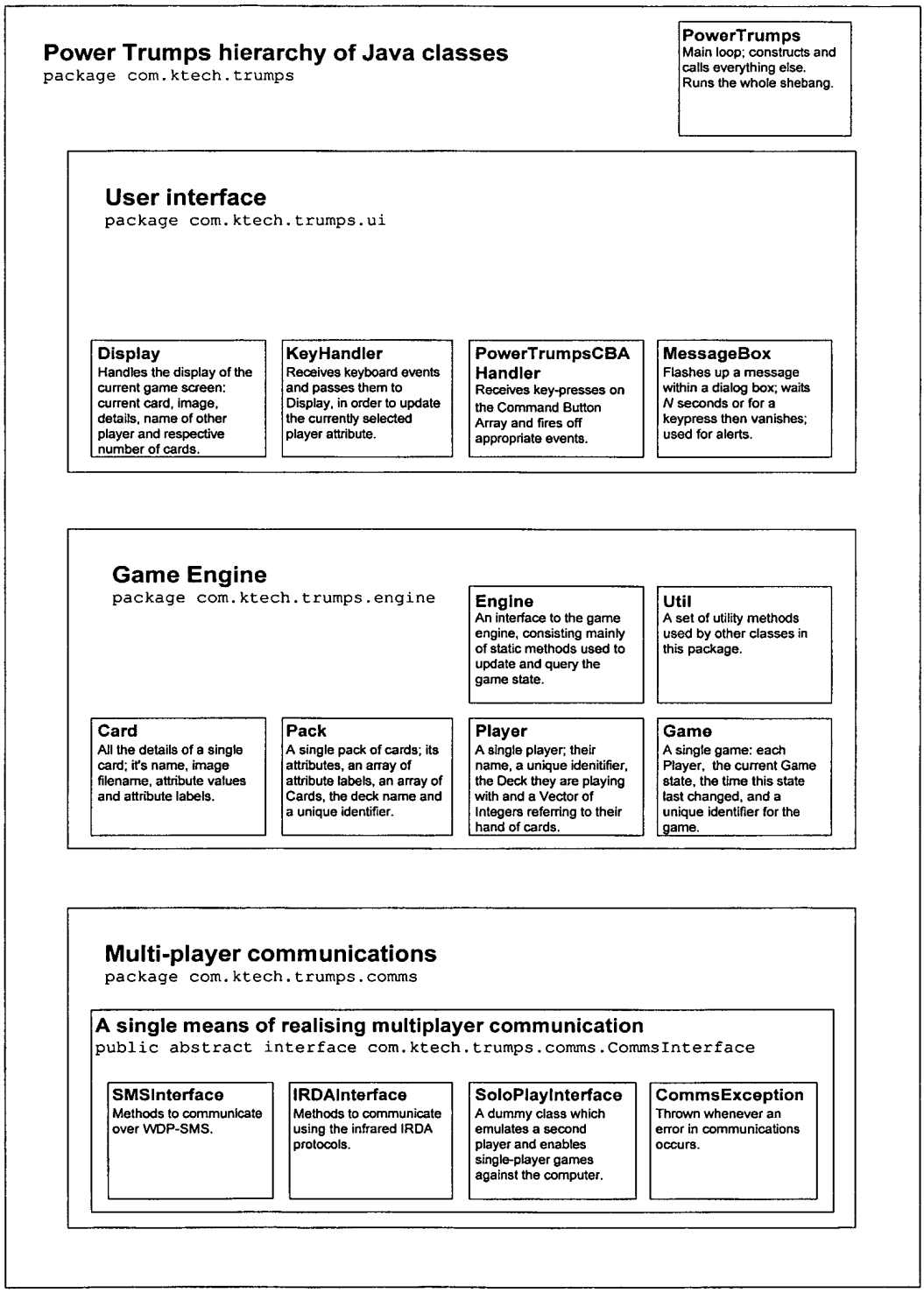

Power Trumps hierarchy of Java classes
package com.ktech.trumps
PowerTrumps
Main loop; constructs and calls everything else. Runs the whole shebang.
User interface
package com.ktech.trumps.ui
Display
Handles the display of the current game screen: current card, image, details, name of other player and respective number of cards.
KeyHandler
Receives keyboard events and passes them to Display, in order to update the currently selected player attribute.
PowerTrumpsCBA Handler
Receives key-presses on the Command Button Array and fires off appropriate events.
MessageBox
Flashes up a message within a dialog box; waits N seconds or for a keypress then vanishes; used for alerts.
Game Engine
package com.ktech.trumps.engine
Engine
An interface to the game engine, consisting mainly of static methods used to update and query the game state.
Util
A set of utility methods used by other classes in this package.
Card
All the details of a single card; it's name, image filename, attribute values and attribute labels.
Pack
A single pack of cards; its attributes, an array of attribute labels, an array of Cards, the deck name and a unique identifier.
Player
A single player; their name, a unique idenitifier, the Deck they are playing with and a Vector of Integers referring to their hand of cards.
Game
A single game: each Player, the current Game state, the time this state last changed, and a unique identifier for the game.
Multi-player communications
package com.ktech.trumps.comms
A single means of realising multiplayer communication
public abstract interface com.ktech.trumps.comms.CommsInterface
SMSInterface
Methods to communicate over WDP-SMS.
IRDAInterface
Methods to communicate using the infrared IRDA protocols.
SoloPlayInterface
A dummy class which emulates a second player and enables single-player games against the computer.
CommsException
Thrown whenever an error in communications occurs.

Figure 2

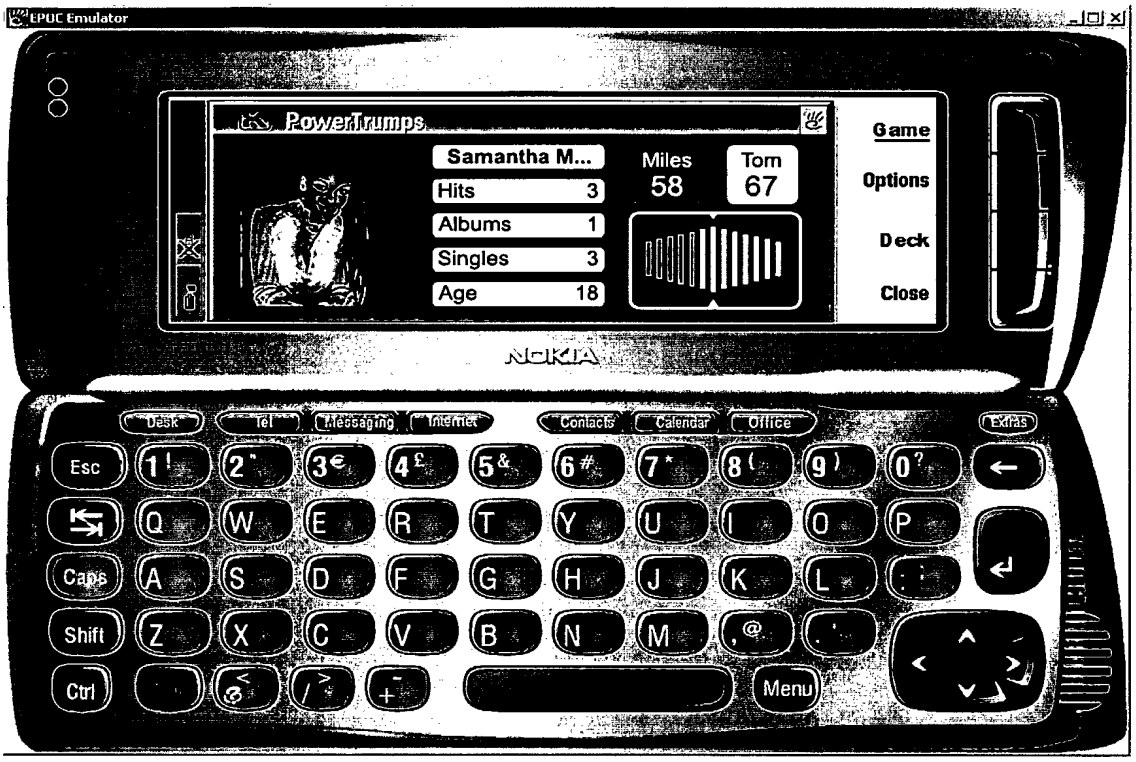

EPOC Emulator
PowerTrumps
Samantha M...
Hits 3
Albums 1
Singles 3
Age 18
Miles 58
Tom 67
Game
Options
Deck
Close
NOKIA
Desk
Tel
Messaging
Internet
Contacts
Calendar
Office
Extras
Esc
Caps
Shift
Ctrl
Menu

Figure 3

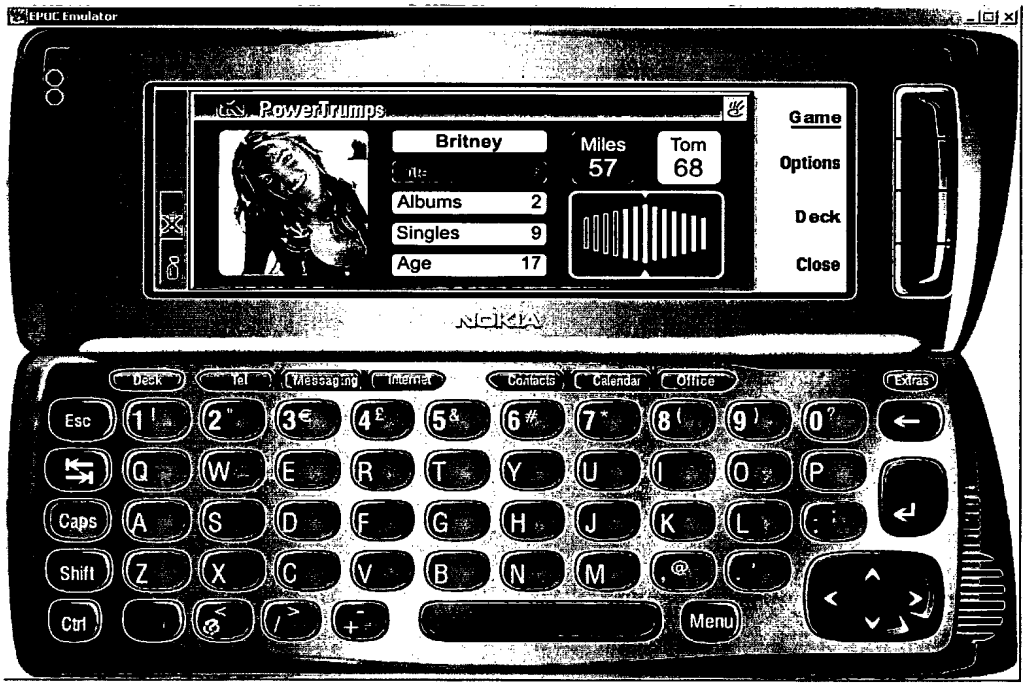

EPOC Emulator
PowerTrumps
Britney
Miles
57
Tom
68
Albums 2
Singles 9
Age 17
Game
Options
Deck
Close
NOKIA
Desk
Tel
Messaging
Internet
Contacts
Calendar
Office
Extras
Esc
Caps
Shift
Ctrl
Menu

Figure 4


PowerTrumps
Britney Spears
Miles
72
Hits: 7
Singles: 8
Albums: 3
Age: 17
Waiting for Miles...

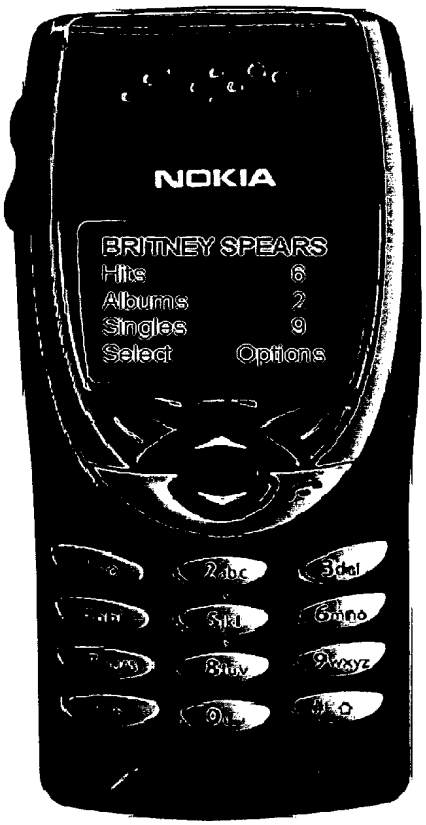

NOKIA
BRITNEY SPEARS
Hits 6
Albums 2
Singles 9
Select Options

Figure 5